Recitation Session, November 15th, 2017

Please do not write on this sheet of paper
And do not use laptops during the session

Exercise 1

Consider the following series:

```
1
1 1
2 1
1 2 1 1
1 1 1 2 2 1
3 1 2 2 1 1
...........
```

1) Find the next element in the sequence above.

```
1 3 1 1 2 2 2 1
```

Now, let us encode an element of the sequence above as a `List[Int]`.

2) Write a function to compute the next element.

```
def nextLine(currentLine: List[Int]): List[Int] = {
  currentLine.foldLeft(List.empty[Int]) { (acc, x) =>
    acc match {
      case y :: count :: rest if x == y => x :: (count + 1) :: rest
      case _ => x :: 1 :: acc
    }
  }.reverse
}
```

3) Implement a stream `funSeq` which constructs this sequence. Recall: to construct a stream, you can use `Stream.cons[A](a: A, b: => Stream[A]): Stream[A]`

```
lazy val funSeq: Stream[List[Int]] =
  Stream.cons(List(1), funSeq.map(nextLine))
```

Exercise 2

1) Write a stream of squares of integers ≥ 1. You may use `Stream.from(i: Int)`

```
val squares: Stream[Int] = Stream.from(1).map(x => x * x)
```

2) Write a stream of all non-empty strings using the characters "0" and "1" and the concatenation operation +. In other words, every non-empty string composed of "0" and "1" should be reached at some point.

```
lazy val codes: Stream[String] = "0" #:: "1" #:: codes.flatMap {
  (s: String) => (s + "0") #:: (s + "1") #:: Stream.empty[String]
}
```

3) Using `codes`, write a stream of all possible non-empty palindromes of “0” and “1”. You may use the `.reverse` function defined on strings.

```
def isPalindrome(x: String): Boolean = x.reverse == x
val palCodes: Stream[String] = codes.filter(isPalindrome)
```

4) Can you do the same without filtering? The palindromes need not to be in the same order.

```
val palCodes: Stream[String] = for {
  c <- codes
  middle <- Seq("", "0", "1")
} yield c + middle + c.reverse
```

5) Given another stream `otherCodes`, possibly finite or infinite, you don’t know at first. Can you build a stream `allCodes` interleaving `palCodes` and `otherCodes` ?

```
def interleave[A](xs: Stream[A], ys: Stream[A]): Stream[A] =
  (xs, ys) match {
    case (x #:: xr, y #:: yr) => x #:: y #:: interleave(xr, yr)
    case (Stream.Empty, _) => ys
    case (_, Stream.Empty) => xs
  }

val allCodes = interleave(palCodes, otherCodes)
```